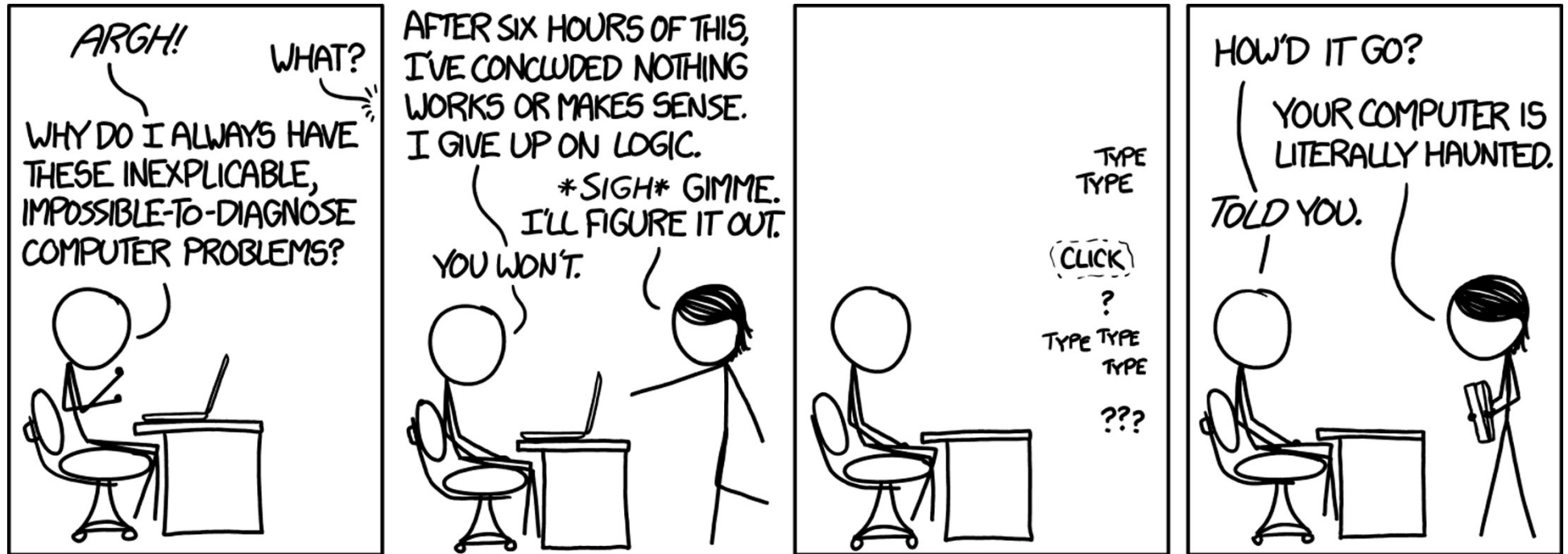




CPEN 311: Digital Systems Design

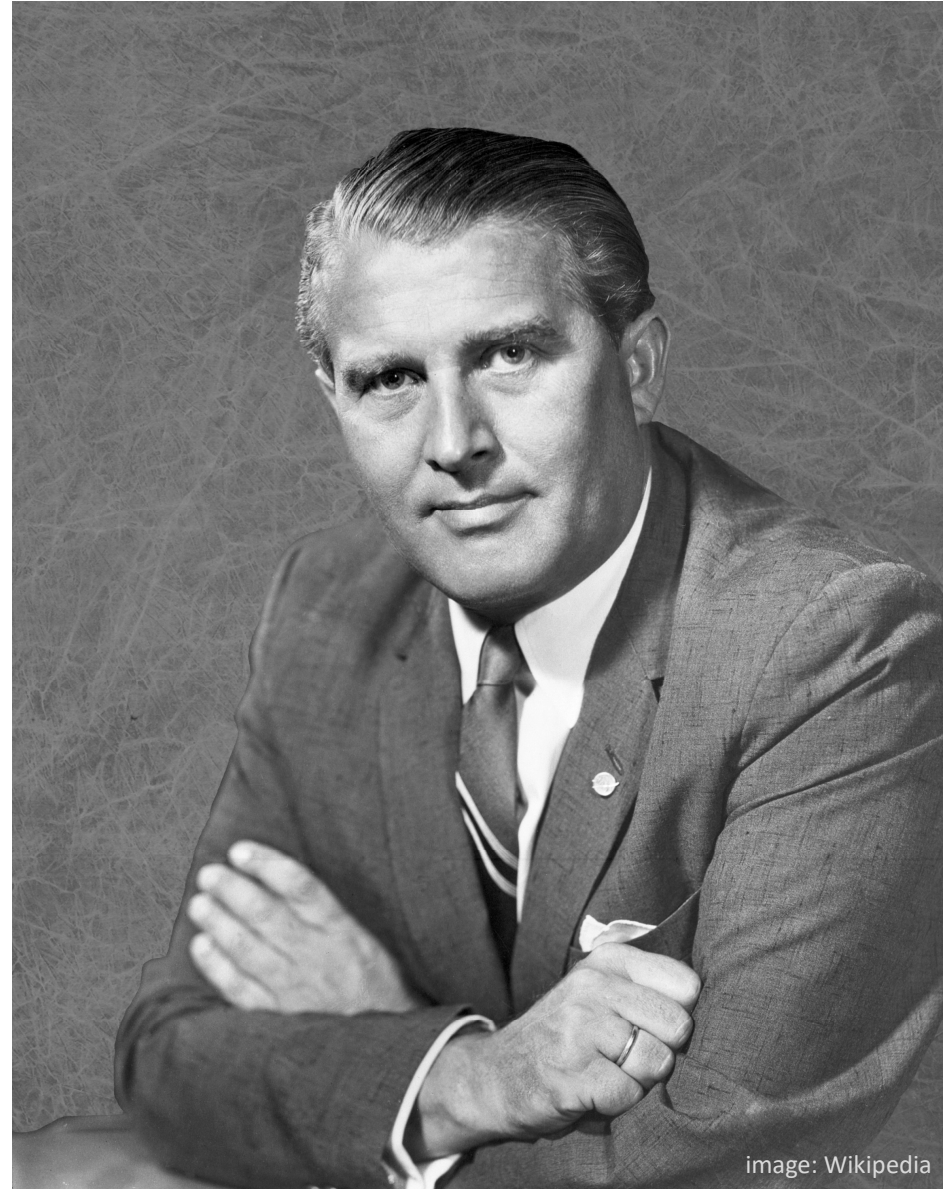
Effective Debugging



xkcd #1316

some thoughts on effective debugging

“failures are good.
you always
learn from them.”



rule #0



rule #1: the red stuff is evil

- undriven/conflicting signals are shown in **RED** in the waveform
- add all signals in design and simulate for a bit, check if stuff is red
- track down and fix all the red stuff
 - if unexpected then you will need to fix it at some point anyway
 - might as well do it now
- probably the **highest payoff/effort ratio**

rule #2: know exactly what to expect


- which exact pixels **should** be drawn? in which order?
 - if you do not know, how will you know when exactly your design fails?
 - a python/C/whatever reference implementation takes 5–10 minutes
 - if you don't know what your design should do, how will you know it works?
- which exact pixels does **your implementation** draw? in which order?
 - `$display`, `$monitor`, `$time`, `diff` \leftarrow all of these should be your **good friends**
 - E.g., print pixel coordinates sequence and compare to “known good” result
- payoff: you will know exactly in **which cycle** the outputs turn bad

rule #3: reproduce the problem RELIABLY

- a lot of “testbenches” that just drive the clock
 - or you did not even simulate anything
 - **That. Is. Not. A. Testbench.**
- stimulate **all possible paths** through design
- **check the outcome and fail if it's not expected**
 - tip: use hierarchical names: `foo.bar.mywire` etc.
 - `assert` and `$error` are your friends
- testbenches should have **tests for all bugs you found**
 - if you can't prove you fixed it, it ain't fixed



rule #4: stop thinking and look

- this is where you look at the **waveform**
 - instead of speculating what the problem might be
- find the failure you saw (you already know the exact failure time)
- use pitchforks  to follow the bad signal
 - if your signal is wrong, **something is driving it that way!**
 - this way you will find the previous bad signal pretty quickly
 - if you follow the signals until you **will** find the problem



Move to the next rising/falling edge of the signal



Tracks signal in data flow

rule #5: practice defensive coding

- think about **invariants** and assert them!
 - e.g., `x-coord < 160; y-coord < 120`
- faking assertions
 - add a signal for your “assertion”
 - make it trigger an always block w/ `$error()` inside
 - surround with ``ifdef` as necessary
- SV has many advanced assertions but our (free) version of ModelSim doesn't support them ☹

rubber duck debugging

- explain your strategy **in words** (e.g., to TA)
 - not: “I’m 100% sure my FSM works, but ...”
- not your Verilog
- not your datapath diagram
- not your FSM diagram
- the reason this works is that it makes you **think** about what your code is doing



debugging is a lot easier than you think

(if you follow the step-by-step debugging process)